A Fast Track Architecture for UDP/IP and TCP/IP

Roy Chua rchua@eecs.berkeley.edu
MacDonald Jackson trey@cs.berkeley.edu
Marylou Orayani marylou@cs.berkeley.edu

May 9, 1995

Abstract

Poor TCP/IP and UDP/IP implementations have been attributed as a factor to the observed lack of performance of remote applications given the speeds of modern communication networks. We discuss some of the factors that cited as contributors to the protocols' poor performance. We then discuss the fast track architecture for UDP/IP and TCP/IP which improves protocol performance by optimizing for the common rather than the general case and by bypassing the operating system completely thereby reducing any overhead that may be incurred while in the operating system. We then present our implementation of a fast track UDP/IP along with proposals for fast track TCP/IP implementations.

1 Introduction

Many researchers have observed that the performance of remote applications have not kept pace with modern communication network speeds. Part of this imbalance is attributed to the protocols used by the remote applications, namely, UDP/IP and TCP/IP. It is now widely believed that the problems with these protocols are not inherent in the protocols themselves but in their particular implementations [Dalt93, Whet95]. The following factors are cited as contributors to the poor performance of UDP/IP and TCP/IP:

Multiple copying of data This is considered the biggest bottleneck. Receiving a packet requires two copies: one copy from the network interface to the kernel and another one from the kernel to the receiving application. Sending a packet also requires two copies: one from the sending application to the kernel and another one from the kernel to the network interface. Measurements in [Kay93] show that moving packet data takes up 9% of TCP processing time and 16% of UDP processing time.

Protocol layering Layering facilitates protocol design but not implementation. It causes bad buffering decisions which generate unnecessary memory traffic [Jacob93]. Packets are forced to go from application buffers to socket buffers to kernel buffers and finally to the network interface buffers. Layering also hinders parallelism [Jacob93] forcing applications to call generic routines instead of protocol-specific ones causing a bottleneck at the generic routines.

Optimization for the general rather than the common case Because of the generality of the code, a lot of time is spent processing packet headers, calculating checksums in the kernel and managing buffers [Whet95]. This was acceptable due to the unreliability of the internetworks within which the protocols were operating but with modern networks, most packets arrive in sequence and are error-free [Jacob93].

Checksum computation It is known that checksum calculation is expensive. In addition, it requires an additional perusal of a packet's header and data. Measurements in [Kay93] show that checksumming of TCP/IP packets take up only 7% of TCP processing time while it takes up 25% of UDP processing time. This is attributed to the larger sizes of UDP packets. Analysis of a FDDI network traffic in a university LAN [Kay93] calculated a median TCP packet size of 32 bytes and a median UDP packet size of 128 bytes.

Kernel buffer manipulation The kernel buffers used for network packet processing are the memory buffers (mbufs) and the socket buffers. According to [Kay93], mbuf and socket buffer processing take up 18% of TCP processing time and 15% of UDP processing time.

Context switches A context switch involves a process blocking itself (e.g., via the `sleep` system call), waking up another process and restarting it. According to Kay et. al. [Kay93], context switches take up about 5% of TCP processing time and 4% of UDP processing time [Kay93].

Error checking Error checking time is largely taken up by the validation of arguments to system calls. This takes up 5% of TCP processing time and 7% of UDP processing time.

Interrupts When a packet arrives, two interrupts are generated: a hardware interrupt generated by the network interface to indicate that a packet has arrived and a software interrupt generated by the network interface device driver to tell IP that a new packet has arrived. Generating a software interrupt and handling the interrupt by dequeueing the incoming packet and calling IP to handle it takes up 2% of processing time for both TCP and UDP.

In [Whet95], a fast track architecture for UDP/IP and TCP/IP was proposed. Its main aim is to improve protocol performance by optimizing for the common case and by bypassing the operating system (OS) completely, reducing any overhead that may be incurred while in the OS. Given network interface support, the fast track can also limit the copying of data to one. The architecture distinguishes between two types of packets: common case and non-common case (common case criteria are discussed in detail in Section 3) and only processes the common case packets. A packet buffer is shared between the application and the network interface device driver. When a common case packet arrives at the receiving host, the device driver copies the packet into the packet buffer it shares with the application. This effectively demultiplexes the packet directly to the application and bypasses the operating system completely. To send a common case packet, the sending application directly copies the packet into the buffer it shares with the device driver and calls a device driver routine to send the packet to its destination.

The fast track architecture is applicable to a wide range of network interfaces [Whet95] and can coexist with other UDP/IP and TCP/IP implementations.

Section 2 discusses problems reported and solutions proposed by researchers in the area. Section 3 presents the goals of the fast track architecture. Section 4 describes the fast track implementation of UDP/IP along with the current UDP/IP implementation. Section 5 is a discussion of fast track and the current TCP/IP implementation. Section 6 is an analysis of the fast track UDP/IP implementation. Section 7 presents difficulties we encountered during implementation. Section 8 presents our conclusions and summary.

2 Previous Work

Various researchers have proposed and implemented solutions to overcome the problems with UDP/IP and TCP/IP implementations discussed in Section 1. In [Part94], Craig Partridge showed ways to speed up UDP/IP implementation one of which is to calculate the checksum while copying the data. Van Jacobson [Jacob93] has implemented a high performance TCP/IP by reducing the number of interrupts generated for each packet sent or received, by touching any piece of data exactly once, and by doing away with layering and maximizing parallelism. His code only requires 37 instructions and 7 memory references to forward an IP packet, and less than 60 instructions and 22 memory references to process a TCP packet.

Dalton et. al. [Dalt93] modified TCP/IP to support the Afterburner card which they designed and built. The card provides buffers that can be directly accessed by the operating system so that data is copied only once: from the on-card buffers to the application buffers for receiving and vice-versa for sending. They achieved a throughput of 210 Mbps with 14 KB packets.

Brustoloni [Brust94] used exposed buffering to avoid copying of data altogether. A special area in kernel memory is allocated specifically for these exposed buffers so that both the applications and the network interface can share access to them. When a packet is received, it is placed directly in these buffers and the

application reads the packet directly from them. To send a packet, the application writes into the buffers and the network interface sends it from the same buffers. This was implemented in the Mach operating system on a host connected to an ATM LAN. They achieved a throughput of 48.4 Mbps with 64 KB datagrams and 48.5 Mbps with 64 KB sequenced packets.

Kay et. al. [Kay93] suggested avoiding checksumming for both TCP and UDP when the communicating hosts are on the same LAN provided the LAN supports a hardware CRC. They also stated that reducing TCP protocol-specific processing time is useful but will require a major modification of TCP implementation. It is unclear whether the effort required is worth the performance improvement that will be gained.

Whetten et. al. [Whet95] proposed an implementation for a fast track receiver and sender for both TCP/IP and UDP/IP. We initially tried to implement the fast track TCP/IP receiver but encountered difficulties with the proposed implementation. The proposal did not take into account that for both the fast track and non-fast track TCP/IP implementations to coexist, they must be able to share state information about TCP/IP connections. We had a few options:

- Remove the requirement that fast track and non-fast track TCP/IP must coexist.

  This is infeasible since fast track setup requires assistance from non-fast track TCP/IP. In addition, non-common case TCP/IP packets will never be unprocessed.

- Modify TCP/IP to be able to more readily share state information with fast track TCP/IP.

  Given the complexity of TCP/IP code, we believed this would require too much work given the amount of time we had.

- Implement fast track UDP/IP instead.

  This is the option we eventually chose because of the simplicity of UDP/IP relative to TCP/IP.

In the remaining sections of the paper, we first present the fast track goals for both UDP/IP and TCP/IP and follow this with a discussion of fast track UDP/IP architecture and implementation. This is then followed by a discussion of the fast track TCP/IP architecture and a number of proposals for its implementation. We then present an analysis of our fast track UDP/IP implementation.

3 Fast Track Goals

For both UDP/IP and TCP/IP, the goals of the fast track architecture are to optimize for the common case (vs. the general case), to minimize operating system overhead and to minimize data copying. A UDP/IP packet is considered common case for the receiver if it satisfies the following criteria:

- It uses the correct IP version (4).
- It has the correct header length (5).
- It is not fragmented.
- It is destined for this host.
- It is error-free.
- A socket already exists for the packet.

The criteria for a TCP/IP packet are identical with the following additions:

- It is not a control packet.
- It has the correct sequence number (i.e., the packet arrived in order).
- It has no variable length options.

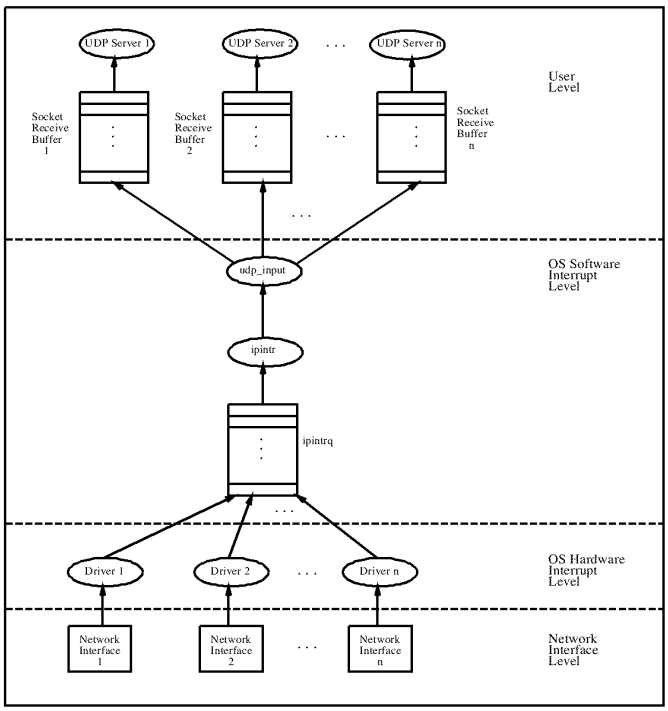

Figure 1: Current UDP/IP Receiver

The criteria for a common case UDP/IP packet for the sender are:

- No packets waiting to be sent.
- No fragmentation required.
- Already have an outgoing socket.
- Have already resolved the outgoing link address using ARP

The criteria for sending a TCP/IP packet are identical with the following additions:

- Window size must be big enough and have space.
- Must have a valid connection

Though treated separately in this paper, fast track UDP/IP and TCP/IP can and should be combined in one implementation. The separation was done for simplicity and clarity.

4 Fast Track UDP/IP

In this section, we will first describe the current implementation of the UDP/IP receiver and then contrast it with the fast track implementation. This is followed by a similar discussion of the UDP/IP sender.

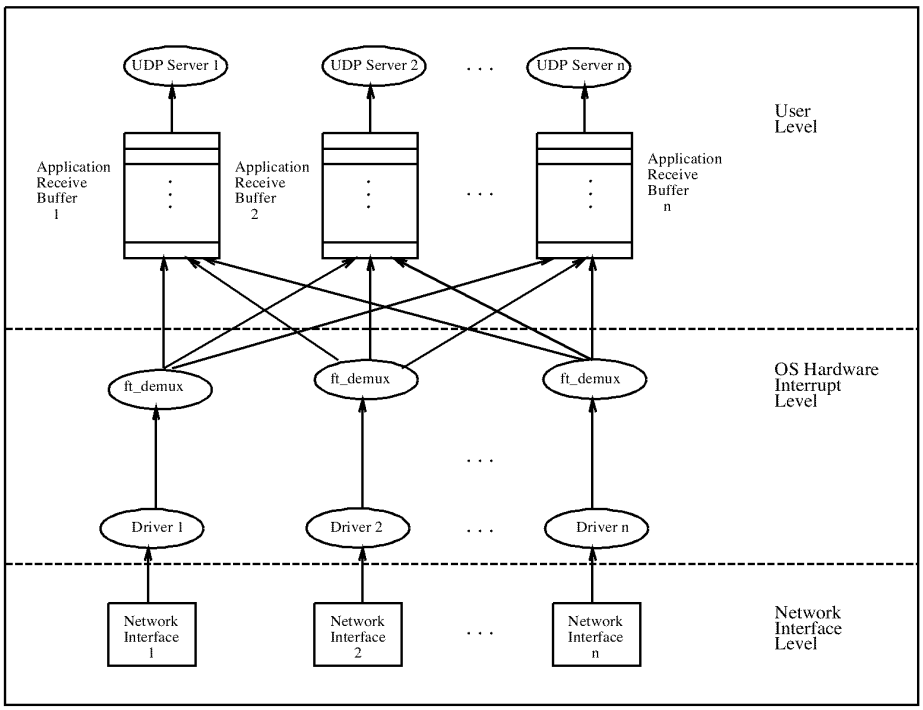

Figure 2: Fast Track UDP/IP Receiver

4.1 Current UDP/IP Receiver

Figure 1 shows a diagram of the current UDP/IP receiver. When a packet arrives at the network interface, it generates a hardware interrupt which activates the interface's device driver. If the packet is an IP packet, the driver puts it into the IP input queue, `ipintrq`, and generates a software interrupt to inform the IP interrupt handler, `ipintr`, that a new IP packet has arrived. Based on the packet's protocol type (specified in the IP header), IP demultiplexes the packet to the appropriate protocol. If it is a UDP packet, `ipintr` invokes the UDP input function, `udp_input`, to process the incoming packet. `udp_input` first validates the incoming packet by verifying its length and by calculating its checksum. `udp_input` then determines the packet's recipient application and puts the packet into that application's socket receive buffer. Two interrupts are generated, the hardware interrupt generated by the network interface and the software interrupt generated by the device driver. One can clearly see the bottleneck at `ipintr` and `udp_input` due to layering.

4.2 Fast Track UDP/IP Receiver

We implemented the receiver on a HP-UX 9000/700 workstation running the HP-UX OS version A.09.01.

Figure 2 shows a diagram of the fast track UDP/IP receiver. When a packet arrives at the network interface, it generates a hardware interrupt which activates the interface's device driver. The driver then calls the fast track demultiplexer, `ft_demux`, which determines if the packet is common case or not. If it satisfies all the common case criteria enumerated in Section 3 above, `ft_demux` copies the packet from the network interface to the appropriate application receive buffer (which replaces the socket receive buffer). Otherwise, the packet is placed in the appropriate protocol input queue and the appropriate software interrupt is generated. Only one interrupt is generated, the hardware interrupt generated by the network interface. Except for the brief stay in the demultiplexer, the operating system is completely bypassed. In addition, no bottlenecks exist.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.